模式识别与深度学习 实验二

一.实验环境

实验平台：google colab
处理器：Intel(R) Xeon(R) CPU @ 2.00GHz
内存：16GB
显卡：Tesla T4
PyTorch 版本：1.9.0

二.实验内容

本次实验旨在使用 PyTorch 框架实现 AlexNet 模型，并在 Caltech101 数据集上训练和评估模型的性能。具体实验内容和过程如下：

2.1 数据准备

我们在官网下载了 Caltech101 数据集，并人为删除 BACKGROUND_Google 文件夹，便于后续分类处理。

编写数据划分函数,将每个类别中的图片划分为训练集、验证集和测试集。

```
for category in os.listdir(data_dir):
    # 定义每个类别的目录
    category_dir = os.path.join(data_dir, category)
    # 如果不是文件夹，则跳过
    if not os.path.isdir(category_dir):
        continue
    # 获取每个类别中所有图片的文件名
    image_names = os.listdir(category_dir)
    # 随机打乱文件名的顺序
    random.shuffle(image_names)
    # 计算训练集、验证集和测试集的数量
    num_images = len(image_names)
    num_train = int(num_images * train_ratio)
    num_valid = int(num_images * valid_ratio)
    num_test = num_images - num_train - num_valid
    # 定义每个集合的目录
    train_category_dir = os.path.join(train_dir, category)
    valid_category_dir = os.path.join(valid_dir, category)
    test_category_dir = os.path.join(test_dir, category)
    # 创建每个集合的目录
    os.makedirs(train_category_dir, exist_ok=True)
    os.makedirs(valid_category_dir, exist_ok=True)
    os.makedirs(test_category_dir, exist_ok=True)
    # 将图片复制到对应的集合目录中
    for i, image_name in enumerate(image_names):
        src_path = os.path.join(category_dir, image_name)
        if i < num_train:
            dst_path = os.path.join(train_category_dir, image_name)
        elif i < num_train + num_valid:
            dst_path = os.path.join(valid_category_dir, image_name)
        else:
            dst_path = os.path.join(test_category_dir, image_name)
        shutil.copyfile(src_path, dst_path)
```

我们将图片 resize 为 224*224 大小，还对训练集进行了数据增强（随机水平翻转），以提高模型的泛化能力。

```python
train_transform = transforms.Compose([
    transforms.Resize((224, 224)), # 将图像大小调整为224x224
    transforms.RandomHorizontalFlip(), # 随机水平翻转
    transforms.ToTensor(), # 将图像转换为张量
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)), # 标准化
])
```

2.2 模型实现

我们使用 PyTorch 框架实现了 AlexNet 模型，该模型包含特征提取部分和分类部分两个主要部分。特征提取部分由五个卷积层和三个最大池化层组成，用于提取输入图像的特征。分类部分由三个全连接层和两个 Dropout 层组成，用于将特征映射到类别上。AdaptiveAvgPool2d 层用于将特征图的大小调整为固定大小，以便将其输入到全连接层中。在前向传播过程中，输入图像首先通过特征提取部分，然后通过平均池化层，最后通过分类部分，得到最终的分类结果。

```python
# 定义模型
class AlexNet(nn.Module):
    def __init__(self, num_classes=101):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

2.3 模型训练

在训练过程中，使用交叉熵损失函数以及 SGD 优化器对模型进行训练，初始学习率为 0.01，权重衰减因子为 0.0001。我们训练了 50 个 epoch，并在每个 epoch 结束后在验证集上评估模型的性能。

首先定义了两个变量 best_val_loss 和 best_val_acc，分别用于保存在验证集上表现最好的模型的损失值和准确率。这些变量的初始值分别为正无穷和 0.0。

接着，代码通过一个循环结构来进行训练和验证。在每个循环周期内，首先调用 train 函数对模型进行训练，然后调用 validate 函数对模型进行验证，并计算验证集上的损失值和准确率。

如果当前模型的验证损失值比之前最好的模型的损失值还要小，那么就将当前模型的参数保存到磁盘上，以便后续使用。如果当前模型的验证准确率比之前最好的模型的准确率更高，那么就更新 best_val_acc 变量的值。

如果验证集上的损失值连续多次超过之前最好的模型的损失值的 1.2 倍，那么就停止训练，因为模型有可能发生了过拟合。

```
# 训练模型并验证
best_val_loss = float('inf')
best_val_acc = 0.0

for epoch in range(1, 51):
    train(model, train_loader, criterion, optimizer, epoch, device, writer)
    val_loss, val_acc = validate(model, val_loader, criterion, device, writer)

    # 保存在验证集上表现最好的模型
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 'best_model.pth')

    if val_acc > best_val_acc:
        best_val_acc = val_acc

    # 如果验证损失值连
    if val_loss > best_val_loss * 1.5:
      print('Validation loss is not improving, stop training.')
      break
```

2.4 模型评估

我们使用在验证集上表现最好的模型在 test 集上评估模型的性能。

```python
model.load_state_dict(torch.load('best_model.pth'))
model.eval()

test_loss = 0.0
correct = 0
total = 0

for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)

    with torch.no_grad():
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

test_loss /= len(test_loader)
test_acc = 100 * correct / total

print('Test Loss: {:.4f} Accuracy: {:.2f}% ({}/{})'.format(test_loss, test_acc, correct, total))
```

三.实验结果与分析

最佳模型在验证集上的准确率为 72.23%

```
Validation Loss: 1.5065 Accuracy: 72.23%
```

最佳模型在测试集上的准确率为 64.35%

```
Test Loss: 1.7089 Accuracy: 64.35% (612/951)
```

下图为使用 tensorboard 展示训练过程中的 loss 曲线，可以看出在早期 epoch，loss 下降较快，训练效果较好，但中期出现了 loss 的上升。